Курсовая работа

Тагиев Байрам Алтай оглы

Содержание

# 1 Сетевой симулятор - ns-2

## 1.1 Что такое NS-2 и для чего он нужен?

NS-2 (Network simulator 2) - это симулятор дискретных событий, предназначенный для исследования сетей. NS-2 предоставляет существенную поддержку для моделирования протоколов TCP, маршрутизации и многоадресной рассылки по проводным и беспроводным (локальным и спутниковым) сетям.

NS - самый популярный выбор симулятора, используемый в исследовательских статьях, появляющихся на избранных конференциях, таких как Sigcomm. ns постоянно поддерживается и обновляется своей большой базой пользователей и небольшой группой разработчиков в ISI.

Сам по себе NS-2 просчитывает то, что происходит в симуляции, но для наглядности нам нужна визуализация всего процесса. Для этого был создан NAM - Network Animator. NS вместе со своим компаньоном, nam, образуют очень мощный набор инструментов для обучения концепциям сетевого взаимодействия. NS содержит все основные протоколы IP. С помощью NAM эти протоколы можно визуализировать в виде анимации.

## 1.2 Что мы можем сделать с помощью этих инструментов?

Создавать:

1. Наземные, спутниковые и беспроводные сети с различными алгоритмами маршрутизации (DV, LS, PIM-DM, PIM-SM, AODV, DSR).
2. Источники трафика, такие как Web, ftp, telnet, cbr, случайный трафик.
3. Сбои, включая детерминированные, вероятностные потери, сбой связи и т.д.
4. Различные дисциплины организации очередей (drop-tail, RED, FQ, SFQ, DRR и т.д.) и QoS (например, IntServ и Diffserv).

Визуализировать:

1. Поток пакетов, наращивание очереди и отбрасывание пакетов.
2. Поведение протокола: медленный запуск TCP, саморегулирование, контроль перегрузки, быстрая повторная передача и восстановление.
3. Перемещение узлов в беспроводных сетях.
4. Аннотации для освещения важных событий.
5. Состояние протокола (например, TCP cwnd).

# 2 Аналоги

Самые известные и популярные аналоги NS-2:

* NS-3
* Cisco Packet Tracer
* MIMIC Simulator

## 2.1 MIMIC Simulator

MIMIC Simulator - это набор продуктов Gambit Communications, состоящий из программного обеспечения для моделирования в области управления сетями и системами.

Пакет MIMIC Simulator Suite содержит несколько компонентов, связанных с имитацией управляемых сетей и центров обработки данных в целях разработки программного обеспечения, тестирования или обучения программного обеспечения, продаж и маркетинга приложений для управления сетями.

MIMIC SNMP решает классическую задачу моделирования: программное обеспечение системы управления сетью или поддержки операций обычно управляет большими сетями. Традиционно для создания таких сетей для вышеуказанных целей физическое оборудование приобреталось отдельно и монтировалось в лабораториях. Чтобы снизить затраты, большая часть сети может быть смоделирована

## 2.2 Packet Tracer

Packet Tracer - это кроссплатформенный инструмент визуального моделирования, разработанный Cisco Systems, который позволяет пользователям создавать сетевые топологии и имитировать современные компьютерные сети. Программное обеспечение позволяет пользователям моделировать конфигурацию маршрутизаторов и коммутаторов Cisco, используя имитированный интерфейс командной строки. Packet Tracer использует пользовательский интерфейс перетаскивания, позволяющий пользователям добавлять и удалять имитируемые сетевые устройства по своему усмотрению.

Packet Tracer позволяет пользователям создавать имитированные сетевые топологии путем перетаскивания маршрутизаторов, коммутаторов и различных других типов сетевых устройств. Физическое соединение между устройствами представлено элементом “кабель”. Packet Tracer поддерживает множество имитируемых протоколов прикладного уровня, а также базовую маршрутизацию с помощью RIP, OSPF, EIGRP, BGP.

## 2.3 NS-3

NS-3 является прямым наследником NS-2. NS-3 построен с использованием C++ и Python с возможностью написания сценариев. Библиотека NS обернута в Python благодаря библиотеке pybindgen, которая делегирует синтаксический анализ заголовков NS C++ в castxml и pygccxml для автоматической генерации соответствующего связующего элемента C++. Эти автоматически сгенерированные файлы C++ в конечном итоге компилируются в модуль NS Python, чтобы позволить пользователям взаимодействовать с моделями C++ NS и ядром с помощью скриптов Python. Симулятор NS оснащен интегрированной системой на основе атрибутов для управления значениями параметров моделирования по умолчанию и для каждого экземпляра.

# 3 RED

## 3.1 Теоретическое введение

RED ([1]) (Random Early Detection - Произвольное раннее обнаружение) – Алгоритм активного управления очередью для управления переполнением очередей маршрутизаторов, с возможность предотвращения перегрузок.

Вероятность маркировки на отбрасывание пакетов представляет собой функцию, линейно зависящую от , минимального и максимального пороговых значений и параметра , определяющего часть отбрасываемых пакетов при достижении средним размером очереди значения и вычисляется следующим образом:

## 3.2 Сравнение с DropTail

DropTail относится к пассивному типу управления очередью (PQM). Он сохраняет пакет до тех пор, пока буфер не заполнится, и когда буфер заполнится, т.е. в очереди не останется свободного места, он начинает отбрасывать каждый пакет. Существуют две возможные вероятности выпадения, т.е. 0 или 1. Вероятность отбрасывания пакета равна 0, если количество поступивших пакетов меньше количества буферизованных пакетов, и в обратном случае она будет равна 1.

Основным отличием между RED и DropTail является то, что RED позволяет контролировать пропускную способность и более эффективно управлять трафиком, тогда как Droptail не гарантирует равномерное распределение пропускной способности. ([2])

## 3.3 Разбор алгоритм работы RED

Пакет при поступлении в систему попадает в модуль сброса. Решение о сбросе пакета принимается на основе значения вероятности , получаемого от управляющего модуля. Вероятность сброса пакетов зависит от экспоненциально взвешенного скользящего среднего размера длины очереди , также вычисляемого управляющим модулем, основы- ваясь на текущем значении длины очереди .

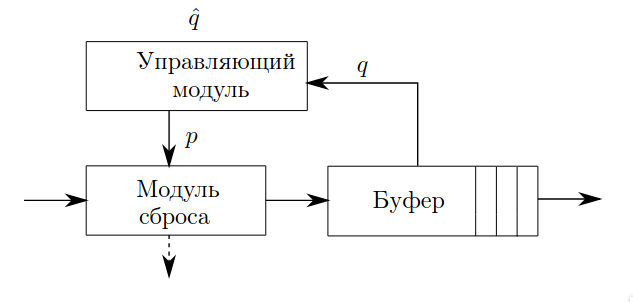


Рис. 1: Модуль RED

Вероятность потери пакета в зависимости от среднего размера очереди будет выглядеть следующим образом:

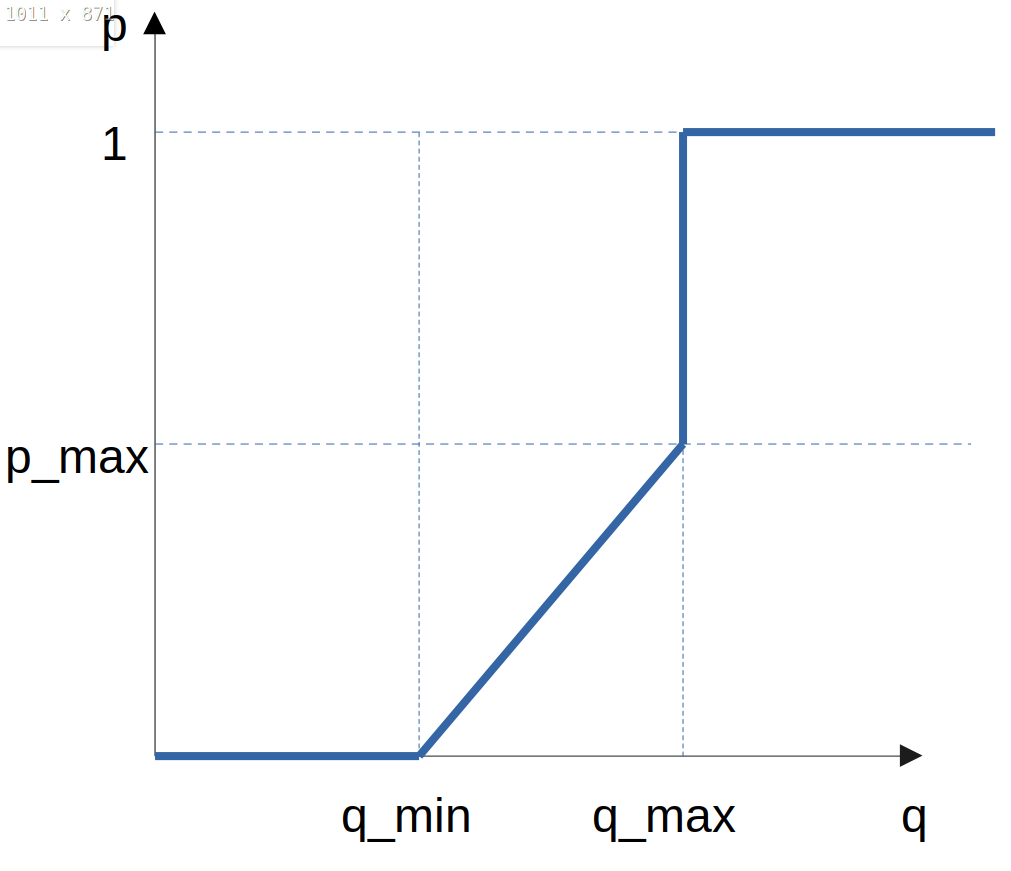


Рис. 2: RED

## 3.4 Разбор реализации в NS2

Файлы, связанные с RED находятся по пути ns-2.35/queue, там представлены различные реализации очередей (среди них DropTail, RED и т.д.). Для нас важны два файла:

* red.h – Заголовочный файл
* red.cc – Исходники

В файле red.cc для нас интересна функция REDQueue::estimator и функция double REDQueue::calculate\_p\_new. Первая функция отвечает за расчет средней длины очереди. Вторая – расчитывает вероятность потери пакета.

Разберем исходный код второй функции:

double   
REDQueue::calculate\_p\_new(double v\_ave, double th\_max, int gentle, double v\_a,   
 double v\_b, double v\_c, double v\_d, double max\_p)  
{  
 double p;  
 if (gentle && v\_ave >= th\_max) {  
 // Необходимо для GRED (подробнее о GRED ниже)  
  
 // p находится в промежутке от max\_p до 1,  
 // тогда как средний размер очереди в промежутке  
 // q\_max и 2\*q\_max  
 p = v\_c \* v\_ave + v\_d;  
 } else if (!gentle && v\_ave >= th\_max) {   
 // Превысили пороговое значение в классическом RED  
 // p приравниваем к 1  
 p = 1.0;  
 } else {  
 // p в промежутке от 0 до max\_p, тогда как  
 // средний размер очереди в промежутке  
 // th\_min до th\_max  
 p = v\_a \* v\_ave + v\_b;  
 // p = (v\_ave - th\_min) / (th\_max - th\_min)  
  
 p \*= max\_p;   
 }  
 if (p > 1.0)  
 p = 1.0;  
 return p;  
}

## 3.5 Проблемы RED

Одна из фундаментальных проблем RED заключается в том, что он полагается на длину очереди в качестве показателя загруженности. Хотя наличие постоянной очереди указывает на перегрузку, ее длина дает очень мало информации о серьезности перегрузки.

Поскольку алгоритм RED зависит от длины очереди, ему присуща проблема определения степени перегрузки. В результате RED требует широкого диапазона параметров для корректной работы в различных сценариях перегрузки. Хотя RED может достичь идеальной рабочей точки, он может сделать это только при наличии достаточного объема буферного пространства и правильных параметров.

# 4 WRED

## 4.1 Теоретическое введение

WRED (Weighted random early detection - Взвешенное произвольное раннее обнаружение) – Алгоритм активного управления очередью, является расширением RED.

Взвешенный алгоритм произвольного раннего обнаружения предоставляет различные уровни обслуживания пакетов в зависимости от вероятности их отбрасывания и обеспечивает избирательную установку параметров механизма RED на основании типа трафика.

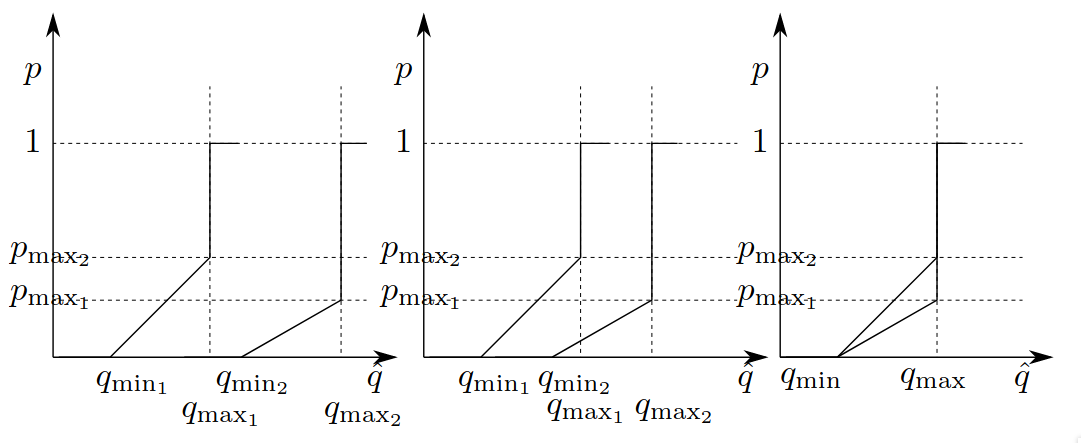


Рис. 3: WRED

Алгоритм WRED работает с единой очередью пакетов, для которой, как и в RED, по формуле рассчитывается экспоненциально взвешенное скользящее среднее. Для каждого типа трафика задаются собственные параметры (пороговые значения, максимальный уровень сброса) и вычисляется вероятность сброса.

Например, очереди могут иметь более низкие пороговые значения для более низких приоритетов пакета. Это приведет к отбрасыванию пакетов с низким приоритетом, а следовательно, к защите пакетов с более высоким приоритетом в той же очереди.

# 5 GRED

## 5.1 Теоретическое введение

GRED (Gentle random early detection - мягкое/аккуратное произвольное раннее обнаружение) – Алгоритм активного управления очередью, является расширением RED.

Gentle RED расширяет RED тем, что добавляет дополнительное максимальное пороговое значние, которое равно , тем самым “сглаживая” кривую.

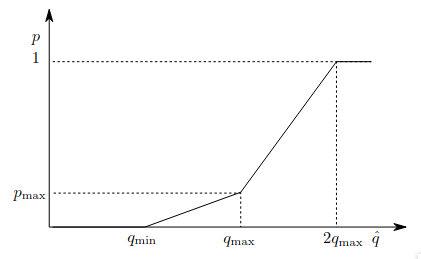


Рис. 4: GRED

Вычисляется следующим образом:

# 6 NLRED

## 6.1 Теоретическое введение

RED не поддерживает состояние каждого потока, то есть данные из всех потоков помещаются в одну очередь и концентрируются на их производительности. Это порождает проблемы, вызванные невосприимчивыми потоками. Для решения этой проблемы был внедрен алгоритм управления перегрузками нелинейного случайного раннего обнаружения (NLRED ([3])). В NLRED есть нелинейная квадратичная функция, где в RED функция отбрасывания линейна.

Вместо линейного увеличения вероятности, возможно, было бы лучше, если бы вероятность увеличивалась медленно, когда она близка к минимальной, и резко увеличивалась, когда она близка к максимальной. Если вероятность отбрасывания увеличивается линейно между минимальным порогом и максимальным порогом, то существует высокая вероятность того, что входящий пакет может быть отброшен, даже если средняя длина очереди невелика.

# 7 Adaptive RED

## 7.1 Теоретическое введение

В алгоритме Adaptive RED (ARED), разработанном Фенгом [4,5] и усовершенствованная Флойдом [6], функция сброса модифицируется посредством изменения по принципу AIMD (Принцип AIMD заключается в том, что увеличение некоторой величины производится путём сложения с некоторым параметром, у уменьшение — путём умножения на параметр.)

Алгоритм ARED функционирует следующим образом. Для каждого интервала interval (параметр) в секундах, если больше целевой (желаемой) и , то увеличивается на некоторую величину 𝛼; в противном случае, если меньше целевой и , то уменьшается в раз:

$$

где

## 7.2 Преимущества

* Автоматическая установка минимального порога (minth). Он устанавливается в зависимости от пропускной способности канала (C) и задержки целевой очереди.
* Автоматическая установка максимального порога (maxth). Он устанавливается в зависимости от значения месяца.
* Автоматическая настройка wq. Он устанавливается в зависимости от пропускной способности канала (C).
* Адаптивная настройка maxp. Он адаптирован в соответствии с текущей средней длиной очереди.

# 8 Refined Adaptive RED

Алгоритм Refined ARED (RARED) является модификацией ARED и предлагает более активно изменять вероятность сброса 𝑝max, чтобы иметь возможность быстрой адаптации к изменяющейся экспоненциально взвешенной скользящей средней длине очереди :

где

$$
q\_{min}+0,48(q\_{max}-q\_{min}) < \hat{q\_t} < q\_{min}+0,52(q\_{max}-q\_{min})\\
\alpha=(0,25\frac{\hat{q}-\hat{q\_t}}{\hat{q\_t}})p\_{max}, \ \beta=1-(0,17\frac{\hat{q}-\hat{q\_t}}{\hat{q\_t}-q\_{min}})
$$

# 9 RED на практике

## 9.1 Реализация на NS2

Теперь перейдем к рассмотрению реализации алгоритма RED на NS2. Для удобства мы будем разделять исходный код программы на модули, а в самих модуляхи реализовывать отдельные моменты, необходимые для моделирования.

### 9.1.1 main.tcl

В данном файле мы задаем симулятор, создаем файлы трассировки и добавляем внешние модули, в которых будет реализовываться основная логика. В конце мы запускаем моделирование.

set ns [new Simulator]  
  
set nf [open out.nam w]  
$ns namtrace-all $nf  
  
source "nodes.tcl"  
source "queue.tcl"  
source "plotWindow.tcl"  
source "modeling.tcl"  
source "finish.tcl"  
  
$ns run

### 9.1.2 nodes.tcl

Следующий файл отвечает за создание и соединение узлов. В нем мы задаем 2 маршрутизатора и 20 узлов на каждом конце, соединяем узлы и маршрутизаторы, устанавливаем определенный тип очереди и расставляем их на анимации.

set node\_(r0) [$ns node]  
set node\_(r1) [$ns node]  
$node\_(r0) color "red"  
$node\_(r1) color "red"  
$node\_(r0) label "red"  
  
set n 20  
  
for {set i 0} {$i < $n} {incr i} {  
 set node\_(s$i) [$ns node]  
 $node\_(s$i) color "blue"  
 $node\_(s$i) label "ftp"  
 $ns duplex-link $node\_(s$i) $node\_(r0) 100Mb 20ms DropTail  
  
 set node\_(s[expr $n + $i]) [$ns node]  
 $ns duplex-link $node\_(s[expr $n + $i]) $node\_(r1) 100Mb 20ms DropTail  
}  
  
$ns simplex-link $node\_(r0) $node\_(r1) 20Mb 15ms RED  
$ns simplex-link $node\_(r1) $node\_(r0) 15Mb 20ms DropTail  
  
$ns queue-limit $node\_(r0) $node\_(r1) 300  
$ns queue-limit $node\_(r1) $node\_(r0) 300  
  
  
for {set t 0} {$t < $n} {incr t} {  
 $ns color $t green  
 set tcp($t) [$ns create-connection TCP/Reno $node\_(s$t) TCPSink $node\_(s[expr $n + $t]) $t]  
 $tcp($t) set window\_ 32  
 $tcp($t) set maxcwnd\_ 32  
 $tcp($t) set packetsize\_ 500  
 set ftp($t) [$tcp($t) attach-source FTP]  
}  
  
$ns simplex-link-op $node\_(r0) $node\_(r1) orient right  
$ns simplex-link-op $node\_(r1) $node\_(r0) orient left  
$ns simplex-link-op $node\_(r0) $node\_(r1) queuePos 0  
$ns simplex-link-op $node\_(r1) $node\_(r0) queuePos 0  
  
  
for {set m 0} {$m < $n} {incr m} {  
 $ns duplex-link-op $node\_(s$m) $node\_(r0) orient right  
 $ns duplex-link-op $node\_(s[expr $n + $m]) $node\_(r1) orient left  
}

### 9.1.3 queue.tcl

В данном файле мы настраиваем мониторы и параметры для алгоритма RED. А именно:

* qlim\_ - отвечает за максимальное и минимальное границы для среднего размера очереди
* thresh\_ - минимальная граница для среднего размера очереди в пакетах;
* maxthresh\_ - максимальноая граница для среднего размера очереди в пакетах;
* q\_weight\_ - вес очереди, используется для вычисления среднего размера очереди
* linterm\_ - вероятность отбрасывания пакета
* drop-tail\_ - вместо механизма randomdrop используется механизм drop-tail в случае переполнения очереди или когда средний размер очереди превосходит maxthresh\_

set windowvstime [open wvst w]  
set qmon [$ns monitor-queue $node\_(r0) $node\_(r1) [open qm.out w]]  
[$ns link $node\_(r0) $node\_(r1)] queue-sample-timeout  
  
set redq [[$ns link $node\_(r0) $node\_(r1)] queue]  
$redq set qlim\_ 75 150  
$redq set thresh\_ 75  
$redq set maxthresh\_ 150  
$redq set q\_weight\_ 0.002  
$redq set linterm\_ 10  
$redq set drop-tail\_ true  
  
set tchan\_ [open all.q w]  
$redq trace curq\_  
$redq trace ave\_  
$redq attach $tchan\_

### 9.1.4 plotWindow.tcl

В данном файле хранится процедура для формирование файла с данными о размере окна TCP.

proc plotwindow {tcpsource file} {  
 global ns  
 set time 0.01  
 set now [$ns now]  
 set cwnd [$tcpsource set cwnd\_]  
 puts $file "$now $cwnd"  
 $ns at [expr $now+$time] "plotwindow $tcpsource $file"  
}

### 9.1.5 modeling.tcl

Здесь мы храним наше модельное время, а именно запускаем ftp и процедуру [plotWindow](#plotWindow.tcl)

for {set r 0} {$r < $n} {incr r} {  
 $ns at 0.0 "$ftp($r) start"  
 $ns at 1.0 "plotwindow $tcp($r) $windowvstime"  
 $ns at 20.0 "$ftp($r) stop"  
}  
  
$ns at 21.0 "finish"

### 9.1.6 finish.tcl

А это у нас завершающая процедура, которая генерирует необходимые файлы для построения графиков, закрывает файлы трассировки и запускает xgraph для отрисовки графиков динамики размера окна TPC и график динамики длины очереди и средней длины очереди.

proc finish {} {  
 global ns nf  
 $ns flush-trace  
 close $nf  
 global tchan\_  
 set awkCode {  
 {  
 if ($1 == "Q" && NF>2) {  
 print $2, $3 >> "temp.q";  
 set end $2  
 }  
 else if ($1 == "a" && NF>2)  
 print $2, $3 >> "temp.a";  
 }  
 }  
  
 set f [open temp.queue w]  
 puts $f "TitleText: RED"  
 puts $f "Device: Postscript"  
  
 if { [info exists tchan\_] } {  
 close $tchan\_  
 }  
  
 exec rm -f temp.q temp.a  
 exec touch temp.a temp.q  
  
 exec awk $awkCode all.q  
  
 puts $f \"queue  
 exec cat temp.q >@ $f  
 puts $f \n\"ave\_queue  
 exec cat temp.a >@ $f  
 close $f  
  
 exec xgraph -bb -tk -x time -t "TCPRenoCWND" wvst &  
 exec xgraph -bb -tk -x time -y queue temp.queue &  
 exec nam out.nam &  
 exit 0  
}

### 9.1.7 plot.sh

Чтобы отрисовывать графики на GNUPlot (см. [7,8]) напишем скрипт, который будет доставать данные из наших файлов и подставлять их в графики. Здесь мы задаем границы по оси абсцисс мы задаем промежуток. И отрисовываем три отдельных графика:

* график динамики длины очереди.
* график динамики средней длины очереди.
* график динамики размера окна TCP.

!/usr/bin/gnuplot -persist  
set xrange [0:20]  
  
set terminal postscript eps  
set output "queues.eps"  
set xlabel "Time (s)"  
set ylabel "Queue Length"  
set title "RED Queue"  
plot "temp.q" with lines linestyle 1 lt 1 lw 2 title "Queue length"  
  
set terminal postscript eps  
set output "ave\_queues.eps"  
set xlabel "Time (s)"  
set ylabel "Queue Length"  
set title "RED Queue"  
plot "temp.a" with lines linestyle 2 lt 3 lw 2 title "Average queue length"  
  
set terminal postscript eps  
set output "TCP.eps"  
set xlabel "Time (s)"  
set ylabel "Window size"  
set title "TCPVsWindow"  
plot "wvst" with lines linestyle 1 lt 1 lw 2 title "WvsT"

### 9.1.8 Результат

После выполнения моделирования при помощи ns main.tcl, а также запуска скрипта [plot.sh](#plot.sh) мы получим на выходе три файла формата EPS (Encapsulated PostScript), которые мы можем спокойно просматривать.

* изменение размера длины очереди

![](data:application/pdf;base64,)

* изменение размера средней длины очереди

![](data:application/pdf;base64,)

* изменение размера окна, так как мы задали потолок окна, то он его не будет превышать.

![](data:application/pdf;base64,)

# 10 Варианты RED на практике

## 10.1 NLRED

Для добавления NLRED в ns2 воспользуемся [патчем](http://mohittahiliani.blogspot.com/2012/01/nonlinear-red-nlred-patch-for-ns-2.html) от Mohit P. Tahiliani для версии 2.34.

Изменим файл [queue.tcl](#queue.tcl) для того, чтобы запустить NLRED, добавив в него строчку $redq set nonlinear\_ 1. После того, как мы запустим моделирование, мы получим графики изменения размера длины очереди.

* изменение размера длины очереди

![](data:application/pdf;base64,)

* изменение размера средней длины очереди

![](data:application/pdf;base64,)

## 10.2 Refined Adaptive RED

Для добавления RARED в ns2 воспользуемся [патчем](http://mohittahiliani.blogspot.com/2012/01/refined-adaptive-red-re-ared-or-rared.html) от Mohit P. Tahiliani для версии 2.34.

Изменим файл [queue.tcl](#queue.tcl) для того, чтобы запустить RARED, добавив в него строчку

$redq set adaptive\_ 1  
$redq set refined\_adaptive\_ 1

После того, как мы запустим моделирование, мы получим графики изменения размера длины очереди.

* изменение размера длины очереди

![](data:application/pdf;base64,)

* изменение размера средней длины очереди

![](data:application/pdf;base64,)

# Список литературы

1. Floyd S., Jacobson V. [Jacobson, V.: Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking 1, 397-413](https://doi.org/10.1109/90.251892) // Networking, IEEE/ACM Transactions on. 1993. Т. 1. С. 397–413.

2. Rastogi S., Zaheer H. [Comparative analysis of queuing mechanisms: Droptail, RED and NLRED](https://doi.org/10.1007/s13278-016-0382-5) // Social Network Analysis and Mining. 2016. Т. 6.

3. Rastogi S., Zaheer H. [Comparative analysis of queuing mechanisms: Droptail, RED and NLRED](https://doi.org/10.1007/s13278-016-0382-5) // Social Network Analysis and Mining. 2016. Т. 6.

4. Feng W. и др. Techniques for Eliminating Packet Loss in Congested TCP/IP Networks. 1999.

5. Gateway S. и др. Self-configuring RED gateway // Proceedings - IEEE INFOCOM. 2000. Т. 3.

6. Floyd S., Gummadi R., Shenker S. Adaptive RED: An Algorithm for Increasing the Robustness of RED’s Active Queue Management. 2001.

7. [// gnuplot homepage](http://www.gnuplot.info/).

8. Wikipedia. Gnuplot — Wikipedia, The Free Encyclopedia. <http://en.wikipedia.org/w/index.php?title=Gnuplot&oldid=1149690488>, 2023.